Part 1: (qus 9)

Explanation part:

Given arr[]={ 7, 5, 2, 1, 8, 6, 3, 4} using quicksort for sorting this array ascending order

Now I discuss about quicksort steps

First pass:

Here, low=0, high=n-1

Pivot=arr[high]which is 4;

Int i=0

For(int j=low,j<high-1;j++)

(step according to for loops)

step 1:

J=0, since arr[j]>pivot,do nothing

i=0

step 2:

j=1, since arr[j]>pivot,do nothing

i=0

step 3:

j=2, since arr[j]<=pivot, do i++ and swap(arr[i],arr[j])

arr[]={2,5,7,1,8,6,3,4}//7 and 2 swap

i=1

step 4:

j=3, since arr[j]<=pivot, do i++ and swap(arr[i],arr[j])

arr[]={2,1,7,5,8,6,3,4}//1 and 5 swap

i=2

step 5:

J=4, since arr[j]>pivot,do nothing

i=2

step 6:

J=5, since arr[j]>pivot,do nothing

i=2

step 7:

j=6, since arr[j]<=pivot, do i++ and swap(arr[i],arr[j])

arr[]={2,1,3,5,8,6,7,4}//7 and 3 swap

i=3

loop terminate,now we swap(arr[i],arr[high])//as a result pivot 4 and 5 swapped

At first pass,

we got arr[]={2,1,3,4,8,6,7,5}

Second pass:

Now we divide array into two halves

left part arr[]={2,1,3}

right part arr[]={8,6,7,5}

Then we sort two halves using recursion and swapping return to its main array

Left part become arr[]={1,2,3} sorted order

Right part become arr[]={5,6,7,8} sorted order

Then main array become arr={1,2,3,4,5,6,7,8}

Code part quick sort:

```
#include<bits/stdc++.h>
using namespace std;
int partion(int arr[],int s,int e){
int pivot=arr[e];
int pindex=s;
for(int i=s;i<=e-1;i++){
  if(arr[i]<=pivot){
   swap(arr[i],arr[pindex]);
   pindex++;
  }
}
swap(arr[e],arr[pindex]);
return pindex;
}
void quicksort(int arr[],int s,int e){
if(s<e){
 int p=partion(arr,s,e);
 quicksort(arr,s,p-1);
 quicksort(arr,p+1,e);}
}
int main(){
int n;
cin>>n;
int arr[n];
for(int i=0;i<n;i++)cin>>arr[i];
quicksort(arr,0,n-1);
for(int i=0;i<n;i++)cout<<arr[i]<<" ";
}
```

Part 2 (question 1):

In this problem,unbound knapsack is the best option.Because, sumon can choice any datapack twice or more times.As a result,sumon can buy low cost data packs if he choice best data pack which is value for money.

price[]= {60,150,15,35,120};

datapacks[]={21, 34, 4, 13, 42};

sumon can buy 21 gb twotime,13gb time onetime,4gb datapack onetime.So total=21*2+13+4=170tk.

Otherwise,sumon can buy 42 gb onetime,13gb time onetime,4gb datapack onetime.

so,total cost=120+35+15=170tk

Code part:

```
#include<bits/stdc++.h>
#define INF 1000000
using namespace std;
int main(){
 int val[]={60,150,15,35,120};
    int wt[]={21,34,4,13,42};
            int n = 5,W=59;
            int min_cost[n+1][W+1];
            for (int i=0; i<=W; i++)
            min_cost[0][i] = INF;
            for (int i=1; i<=n; i++)
                        min_cost[i][0] = 0;
            for (int i=1; i<=n; i++){
                        for (int j=1; j<=W; j++){
                                    if (wt[i-1] > j)
                                    min_cost[i][j] = min_cost[i-1][j];
                                    else
                                                min_cost[i][j] = min(min_cost[i-1][j],
                                                            min_cost[i][j-wt[i-1]] + val[i-1]);}
}
cout<< min_cost[n][W]<<endl;}
```

Part 2 (question 2):

Explanation part:

On this question I need to sort all the content in a computer harddisk which is almost full and cannot use extra memory.In this situation between merge sort and selection sort we need selection for sorting this content.Now we discuss why I chose selection sort for this operation.

Selection sort	Merge sort
1. Selection Sort operates in a very simple way. It goes through all the elements in a data-set one by one comparing the value of one element to the next checking to see if the element is smaller, it then saves the smallest element found in a variable and when the iteration is complete, it will insert that saved element to its respective position in that data-set swapping positions with the value in that position.	1.Merge Sort is a divide and conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.So,it creates multiple subset.
2.Selection sort faster than merger sort on small input arrays.	2.On the other hand,Mergesort is not useful for small input arrays.
3.selection sort is a In-Place algorithm which requires short amount of space.	3.Merge sort is not a In-Place algorithm.so it requires O(n) space.
4.Selection sort can be implemented with O(1) auxiliary space.So it doesnot need extra space for sorting array.	4.Merge sort can be implemented with O(n) auxiliary space.So it needs extra space for sorting array.
5.Time complexity:O(n^2) which cannot sort big range of number.	5. Time complexity:O(nlogn)which helps sort big range of numbers

Above this comparasion,I need selection sort because our problem doesnot give extra space for sorting all content.Selection sort can sort input array without any extra spaces because it only compare it next smallest element then swap those smallest element and place array front position according to sorting order.

Part 2(Question 3):

Explanation part:

Companies	1	2	3	4	5
Time(hour)	5	7	6	4	3
Total Taka	32000	30000	21000	36000	21000
Taka/hour	6400	4285.71	3500	9000	7000

Imagine that,there are 5 companies.Maximum time in this festival run 8 hours.Each company offers their best offer.Now,i can discuss using tablation method.

If i sort Taka/hour,then I got taka/hour[]={9000,7000,6400,4285.71,3500)

Total time in this festival w=8hour

At first, programmer select company 4 works according to sorting order and finish full work total_profit=36000TK and w=8-4=4hour.

Then,select company 5 and finish full work total_profit=36000+21000=57000TK and w=4-3=1hour

At last,select company 1 and decide to work for 1 hours total_profit=57000+(1*6400)=63400TK

and w=1-1=0 hour

so,loop terminated total max profit=63400TK can earned.

Code part:

```
#include<bits/stdc++.h>

using namespace std;

#define ll      long long int

#define forn(i,n)    for (int i = 0; i < n; i++)

typedef struct {
```

```
  ll value;
   ll time;
   double fraction;}knapsack;
bool compare(knapsack a,knapsack b){
return (a.fraction > b.fraction);
}
int main(){
ll w,n;
cin>>n;
cin>>w;
knapsack arr[n];
forn(i,n){
cin>>arr[i].time;}
forn(i,n){
cin>>arr[i].value;}
double total=0,currweight=0;
forn(i,n){
arr[i].fraction=(double)arr[i].value/(double)arr[i].time;}
sort(arr, arr+n,compare);
  forn(i,n){
   if(currweight + arr[i].time<= w){
     total += arr[i].value ;
     currweight += arr[i].time;  }
else {
   double wt = w-currweight;
     total += (wt * (double)arr[i].fraction);
     currweight += wt;
     break; }}
cout<<total<<endl;}
```

Part 2: (qus 6)

Explanation part:

Step 1: At first implement sieve of eratosthenes and store all prime numbers in a vector v

Step 2:call sieve of Eratosthenes from main function and check 1 to n number which is divisible by prime numbers.

Step 3: if number divisible by prime numbers we follow two cases.

First case:counter value will increase by 1. if number/prime number=prime number

Second case:else counter value will increase by 2.

Step 4: if (counter +2 ===3)then we can store almost prime number in a vector.

Step 5:Then print almost prime number vector.

Code part:

```
#include <bits/stdc++.h>
using  namespace std;
const int MAX=1e5+10;
vector<bool>prime;
vector<int>v;
void seive(){
prime.assign(MAX,true);
for(int i=2;i*i<=MAX;i++){
 if(prime[i]){
 for(int j=i*i;j<=MAX;j+=i){
  prime[j]=false;
 }
 }
}
for(int i=2;i<=MAX;i++){
  if(prime[i])v.push_back(i);
```

```
}
}
int main(){
 ios_base::sync_with_stdio(false);
   cin.tie(NULL);
   cout.tie(NULL);
seive();
int n;
cin>>n;
vector<int>r;
int c2=0;
for(int i=1;i<=n;i++){
    int c1=0;
  for(int j=0;j<v.size();j++){
   if(i%v[j]==0 ){
       if(i/v[j]==v[j])c1++;
       else c1+=2;
   }
  }
  if(c1+2==3)r.push_back(i);
}
for(int i=0;i<r.size();i++)cout<<r[i]<<" ";
}
```

Part 3: (qus 1)

Explanation part:

We can use dynamic problem to solve this problem.

Step 1:Take input using 2d array and also take another array for calculating longest distance.

Step 2 :then run loop O(nxm)times.

Step 3: if row>0 then we check grid element its left side and colum>0 we can check its upper value

Step 4: Now,we calculate its difference upper and left side value.If difference is exactly +1 or -1.second matrix value become dp[i][j]=max(dp[i][j],1+dp[i-1][j])//for left side

and dp[i][j]=max(dp[i][j],1+dp[i][j-1])//for upper side

step 5: Then we trace this grid and print longest snake sequence.

Code part:

```
#include<bits/stdc++.h>

using namespace std;

const int MAX=1e5+1;

int main(){

 ios_base::sync_with_stdio(false);

     cin.tie(NULL);

     cout.tie(NULL);

int n,m;

cin>>n>>m;

int mat[n][m];

for(int i=0;i<n;i++)

 for(int j=0;j<m;j++)

   cin>>mat[i][j];



int dp[n][m];
```

```
int mx=INT_MIN;
int drow,dcol;
 for(int i=0;i<n;i++){
 for(int j=0;j<m;j++){
   dp[i][j] = 1;
 if(i>0 &&abs(mat[i][j]-mat[i-1][j])==1){
   dp[i][j]=max(dp[i][j],1+dp[i-1][j]);
 }
  if(j>0 &&abs(mat[i][j]-mat[i][j-1])==1){
   dp[i][j]=max(dp[i][j],1+dp[i][j-1]);  }
 if(dp[i][j]>mx){
  mx=dp[i][j];
   drow=i;
   dcol=j;  }
 }
 }
 cout<<endl;
 cout<<"Maximum length of this sqeuence:"<<mx<<endl;
 vector<int>v;
 int i=-1,j=-1;
 while(drow!=i || dcol!=j){
  //cout<<mat[drow][dcol]<<" ";
  v.push_back(mat[drow][dcol]);
i=drow,j=dcol;
if(drow>0 && dp[drow][dcol]==1+dp[drow-1][dcol]){
drow--;
}
  else if(dcol>0 && dp[drow][dcol]==1+dp[drow][dcol-1]){
dcol--;}
```

```
 }
 reverse(v.begin(),v.end());
 cout<<"The longest snake sequence is:( ";
for(int i=0;i<v.size()-1;i++){
   cout<<v[i]<<",";
}
cout<<v[v.size()-1]<<")"<<endl;}
```